Mashups and Widget Orchestration

Ahmet Soylu[1,2], Fridolin Wild[3], Felix Mödritscher[4], Piet Desmet[2],
Serge Verlinde[5], Patrick De Causmaecker[1,2]
[1] K.U. Leuven, Department of Computer Science, Kortrijk, Belgium
[2] K.U. Leuven, Interdisciplinary Research on Technology, Education and Communication - IBBT, Kortrijk, Belgium
{Ahmet.Soylu, Patrick.DeCausmaecker, Piet.Desmet}@kuleuven-kortrijk.be
[3] The Open University, Knowledge Media Institute, Milton Keynes, United Kingdom
f.wild@open.ac.uk
[4] Vienna University of Economics and Business, Department of Information Systems,
Vienna, Austria
felix.moedritscher@wu.ac.at
[5] K.U. Leuven, Leuven Language Institute, Leuven, Belgium
Serge.Verlinde@ilt.kuleuven.be

ABSTRACT
The mashup era has emerged in response to the challenge of integrating existing services, data sources, and tools to generate new applications. Mashups are usually realized either through a seamless integration, in which only the resulting application is known by the end-users, or through integration of original applications, data sources, and tools, particularly in terms of widgets, into the same graphical space, in which participating applications and data sources are identifiable by the end-users. The former composes a unified functionality or data presentation/source from the original sources. The latter generates a digital environment in which participating sources exist as individual entities, but the true integration can only be realized through enabling widgets to be responsive to the events happening in each other. We call such an integration widget orchestration. In this paper, we provide a holistic view on the mashup era and a theoretical grounding for widget-based digital environments, we elaborate on key challenges for realizing such environments and (semi-)automatic widget orchestration, and we introduce our solution strategies. We identified following challenges: widget interoperability, user-behavior mining, and infrastructure. We introduce functional interfaces (FWI) for application interoperability, exploit semantic web technologies for data interoperability, and investigate the possibility of employing workflow/process mining techniques, along with Petri nets as a formal ground, for user-behavior mining. We outline a reference platform and architecture, compliant with our strategies, to foster re-usability of widgets and development of standardized widget-based environments. We have implemented a prototype for a Widget-based Personal Learning Environment (WIPLE) for foreign language learning in order to demonstrate the feasibility of our solution strategies, framework, and architecture.

MEDES'11, November 21-24, 2011, San Francisco, USA.

Categories and Subject Descriptors
D.2.12 [**Software Engineering**]: Interoperability; H.5.4 [**Information Interfaces and Presentation**]: Hypertext/Hypermedia - *Architectures*; K.3.0 [**Computers and Education**]: General; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems.

General Terms
Design, Standardization, Theory.

Keywords
Mashups, Widget Orchestration, Embedded Semantics, RDFa, Ontologies, The Semantic Web, Petri nets, Workflow Mining, Personal Environments, Foreign Language Learning.

1. INTRODUCTION
Although the idea of mashups is not new, nowadays it attracts researchers and practitioners more. This is mainly due to the shift and advancements in web technologies, such as Web 2.0, RESTful services, the Semantic Web, widgets etc. [1-3]. The mashup era has emerged in response to the challenge of integrating existing services, data sources, and tools to generate new applications and gained an increasing emphasis due to the ever-growing heterogeneous application market.

On the one hand, we categorize mashups into two types from an end-user point of view: *box type* and *dashboard type* mashups [4]. The former is realized through a seamless integration combining different applications and data sources into a single user experience, in which only the resulting application is known and perceived by the end-users. The latter is realized through integration of original applications and data sources, particularly in terms of widgets, into the same graphical space (e.g., browser), in which participating applications and data sources can be perceived and identified by the end-users.

On the other hand, from a technical point of view, we categorize mashups with respect to the source and integration approach as depicted in Figure 1.

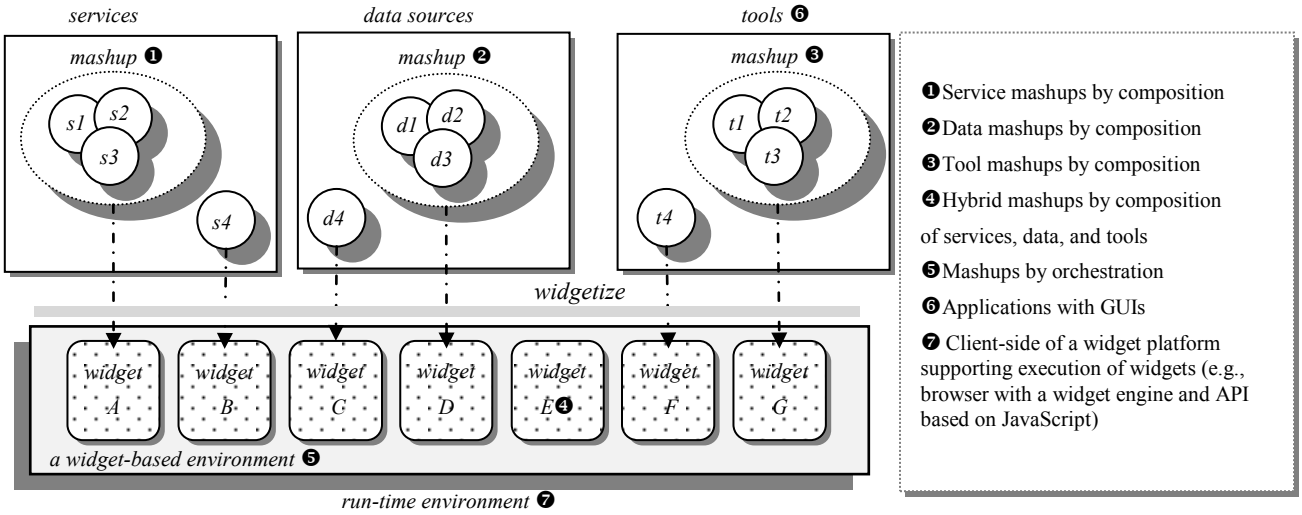

Figure 1. The mashup landscape.

The source-wise categorization includes (1) *service mashups* (e.g., [5]), (2) *data mashups* (e.g., [6]), (3) *tool mashups* (e.g., [7]), and (4) *hybrid mashups* (e.g., [8]). Service and data mashups are based on integration of services and data sources respectively. Tool mashups are similar to the service mashups, however they are based on end-user applications with GUIs, and integration is carried out by extracting and driving the functionality of applications from their user interfaces (e.g., HTML forms). Hybrid mashups combine these three sources.

The integration-wise categorization is linked with and similar to the end-user perspective and includes (1) *mashups by composition* and (2) *mashups by orchestration* (see Figure 2).

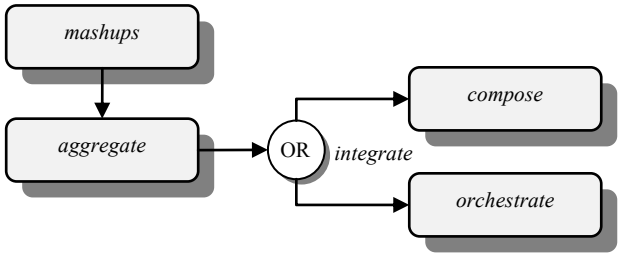

Figure 2. Integration-wise categorization of mashups.

The difference with the end-user perspective lays in its emphasis on the form of functional integration rather than on how the end-users perceive mashups. The former (i.e., by composition) composes a unified functionality or data presentation/source through the seamless functional integration of aggregated sources. The resulting mashup is a new application and participating sources are components of this new application. The latter (i.e., by orchestration) refers to the integration of aggregated resources into the same graphical space in a way that they are functionally and graphically independent of each other. The source applications and data sources are aggregated possibly in terms of widgets through a manual or automated widgetization process. In this respect, if the end application is a widget, an instance of mashup by composition might become an element of an instance of mashup by orchestration. The resulting mashup is a *digital environment* in which participating sources exist as individual entities. The true functional integration can only be realized through enabling widgets to be responsive to the events triggered by other widgets. We call such an integration 'widget orchestration'. Our focus is on (semi-)automatic orchestration, that is enabling a widget platform to learn user behavioral patterns through harnessing event logs and, upon initiation of a pattern, (semi-)automatically executing the corresponding flow, i.e., an ordered set of widget actions. The automation process also requires communication of relevant data through the flow.

In this paper, we provide a global overview on mashups and a theoretical grounding for widget-based (digital) environments, we elaborate on key challenges for realizing such environments and widget orchestration, and we introduce our solution strategies. We have identified the following main challenges: widget interoperability, user-behavior mining, and infrastructure. We introduce functional interfaces (FWI) for widget interoperability, exploit semantic web technologies for data interoperability, and investigate the possibility of employing workflow/process mining, along with Petri nets as a formal ground, for user-behavior mining. We outline a reference platform and its architecture, compliant with our strategies, to foster re-usability of widgets and development of standard widget-based environments. We have implemented a prototype for a Widget-based Personal Learning Environment (WIPLE) for foreign language learning in order to demonstrate the feasibility of our solution strategies.

The rest of the paper is structured as follows. Section 2 describes widgets, widget-based personal environments, and widget orchestration as well as notable challenges. In section 3, the proposed solution strategies are presented along with a platform and architecture. In section 4, an overview of related work is given and the overall approach is evaluated. Finally, section 5 concludes the paper and refers to our future work.

2. WIDGETS AND ORCHESTRATION

The idea of widgets has existed in various forms such as badgets, gadgets, flakes etc., and differentiates with respect to the underlying technology, availability of backend services and so on. In this paper, we are interested in web widgets. Typically, a *web*

widget [9, 10] is a portable, self-contained, full-fledged, and mostly client-side application, hosted online, providing a minimal set of functionality (with/without backend services) with considerably less complex and comprehensive user interfaces. Widgets are expected to be re-usable, which is achieved by enabling widgets to be embedded in different platforms satisfying certain standards and specifications (e.g., W3C widget specifications [10]). Although various technologies can be used to implement widgets (notably HTML and JavaScript, Java Applets, Flash etc.), cross-platform and device support is crucial due to re-usability considerations.

Some widgets are developed for generic purposes such as clock, calendar etc. widgets for specific platforms (e.g., for Windows 7, see Figure 3), mostly, without re-usability and portability concerns. More advanced widgets are developed for specific purposes either from scratch or as a micro version of already existing applications (see Figure 4). As an example of the latter, Figure 4 shows a web application called 'mediatic' (http://www.kuleuven-kortrijk.be/mediatic/ - a foreign language learning tool providing video materials) and its widgetized form which we have developed.

Figure 3. A set of generic purpose widgets for Windows 7.

A widget platform is required to support execution of multiple widgets on a common space. A widget-based environment can be populated by an end-user or pre-populated by a programmer or super-user. Widgets can be populated into a single space, or multiple working spaces can be created to cluster related widgets (e.g., one space for language learning, one for entertainment etc.).

Figure 4. Widgetization of an example application.

During a normal web navigation experience a typical user finds, re-uses, and mixes data by switching between different web applications being accessed either from different browser instances or tabs. Normally, every regular web user generates her own portfolio of applications (implicitly or explicitly, e.g., through bookmarking) over time, hence, for a regular user, one can expect to observe behavioral patterns representing regularly performed actions and activities over the applications in her portfolio (e.g., a user watches videos in a video sharing application and publishes the ones she likes in a social networking application). Widget-based environments facilitate such a user experience by allowing users to access their portfolios through a common graphical space, where each application or data source is represented as a widget, and allows users to create their own *personal environments*. At a conceptual level, a web-based personal (digital) environment [11-13] can be seen as a user-derived sub-ecosystem where member entities come from the Web which is the ultimate *digital ecosystem*. This is because, indeed, the personal environments can be intertwined with the physical world, since the member entities are not limited to web applications and data sources anymore. It includes any physical entity, e.g., devices as well as people, having a *digital presence* [14]. A variety of devices, like mobile phones, tablet PCs, intelligent household appliances, etc. are expected to be connected to the Internet (or to local networks through wired/wireless technologies like Bluetooth etc.) and serve their

functionalities through Web and Web-based technologies, e.g., through RESTful APIs [15]. Hence various Internet-connected devices can be part of the personal environments through widgets allowing users to merge their physical and digital environments into a complete ecosystem (i.e., personal and pervasive) and to organize interactions and data flows between them [8].

In widget-based personal environments, the user experience can be enhanced (1) by enabling data, provided by a user or appearing as a result of her actions in a widget, to be consumable by other widgets, and (2) by automating the execution of regular user actions by learning the user behavioral patterns from logs generated as a result of user actions. We see such interplay between widgets - or web sources in general - as an orchestration process. In general, widget orchestration can happen in various forms in a widget-based environment: (1) *user driven*: the user manually moves data from one widget to the other and initiates the target widget. The manual process can be enhanced through facilitating the data mediation and transportation processes (e.g., select, copy, and paste with drag & drop from widget to widget), (2) *system driven*: the system learns behavioral patterns by monitoring events, each corresponding to a user action, and data emerging as a result, and handles data transportation, mediation, and widget initiation processes (semi-)automatically, (3) *design driven*: a programmer, a super-user or even an end-user pre-codes widget behaviors, e.g., which widget should react to which event and how, and (4) *hybrid*: similar to case 2, the system learns user behavioral patterns, however control is shared, that is instead of going for an immediate automation, the user is provided with recommendations, and the automation only happens if the user decides to follow a particular suggestion. Our focus, in this paper, is on system driven widget orchestration.

However, the widget orchestration faces us with several challenges. The challenges described in what follows are also in the core of the successful realization of other widget orchestration strategies as well as of widget-based environments in general. (1) *Widget interoperability:* (a) *application interoperability* - in order to enable widgets to be responsive to user actions happening in other widgets, a loose functional integration is necessary. Since widgets are developed by different independent parties, standards and generic approaches are required to ensure loose coupling, (b) *data interoperability* - widgets need to share data, particularly during a functional interplay. Since widgets do not have any pre-knowledge about the structure and semantics of data provided by other widgets, standards and generic approaches are required to enable widgets to consume data coming from other widgets. (2) *User behavior mining*: each user action within widgets can be represented with an event. Algorithms that are able to learn user behavioral patterns from the logged events and circulated data are required along with a formal representation paradigm to share, analyze and present behavioral patterns. (3) *Infrastructure:* the abovementioned challenges require any platform, on which the widgets run, as part of possible solution strategies (e.g., how heterogeneous applications communicate events and data). We believe that with the standardization of widget technologies, e.g., widgets, platforms (e.g., run-time system/environment, development frameworks etc.), and reference architectures, widgetization of existing web applications (particularly dynamic approaches) will be of crucial importance due to the growing interest for personal environments. Standardization will enable different communities and parties to develop their own compliant widgets and platforms, and end-users to create their own personal environments by populating heterogeneous applications and data sources, and orchestrating them.

3. SOLUTION STRATEGIES

The overall strategy requires each widget to notify the platform whenever a user action occurs and the platform to store them into the event log. The platform monitors the event log for a certain time and learns behavioral patterns. A behavioral pattern is a partial workflow with a flow structure and routing criteria. Use of domain knowledge, along ontological reasoning support, and standard vocabularies for enhancing event signatures, functional interfaces, widget content and widget interactional elements (e.g., forms) improves the learning process as well as data mediation and transportation. The foremost advantage of our approach is that the widgets and the widget development remain simple, and complicated orchestration tasks are delegated to the platforms.

3.1 Widget Interoperability

Regarding application interoperability, the proposed strategy is that widgets disclose their functionalities through standardized client-sided public interfaces (e.g., JavaScript APIs) which we call *functional widget interfaces* (FWI). As shown in Figure 5, this allows a platform to control widgets through functional interfaces. Each function corresponds to an action that generates an event on each interaction of an end-user. Event notifications and control requests are communicated between the platform and the widgets through a communication channel over a service provided by the run-time system of the platform (see section 3.3 for the platform details). Widgets can share the functionality of their APIs with the platform with a handshake process performed at the first time a widget is added or by extracting it from the event logs. The latter requires functionality provided with GUIs and APIs to be identical. The former is required for a design-driven approach where functionality provided by the widgets should be available to the users (i.e., programmer, super-user, end-user) directly.

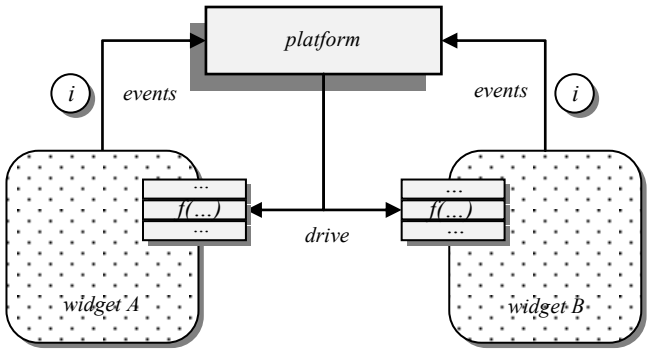

Figure 5. Functional Widget Interfaces (FWI).

An example is given in Figure 6. In this example, there are two widgets in a user's personal environment, namely, 'mediatic' and 'flickr' (a widget that we have developed for a web 2.0 tool that is used to store, sort, search and share photos online – see www.flickr.com). The user watches a video material from the 'mediatic' widget with sub-titles, and when clicking on certain words of the text (the word 'car' in Figure 6), the 'mediatic' widget delivers an event to the platform. The platform decides on an appropriate widget to react on this event. In this case the 'flickr' widget is selected. The relevant event data are extracted and communicated to the 'flickr' widget with the desired

functionality. The 'flickr' widget executes the request by fetching and displaying images relevant to the word of interest.

Figure 6. Widget triggered by an event of another widget.

Concerning the data interoperability, the use of domain knowledge or generic/domain-specific vocabularies enhances the data mediation and transportation. For instance, in Figure 6, the 'mediatic' widget announces an event informing that the noun 'car' is clicked, and the 'flickr' widget is selected to respond although it accepts strings which are of the word type. This is because an ontological reasoning process asserts that the noun 'car' is an instance of the class 'word', since the grounding ontology declares the class 'noun' as a subclass of the class 'word'. A semantic approach also enhances the behavior mining which is described in section 3.2. For this purposes, on the one hand, we enhance events and function signatures with domain ontologies or vocabularies as shown in Figure 7.

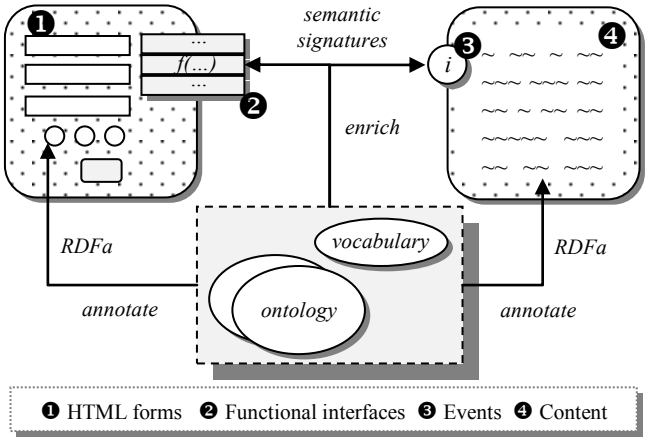

Figure 7. Use of domain knowledge for data interoperability.

On the other hand, we annotate widget content and interactional elements (e.g., forms) with domain knowledge in order to enable end-users to move content from one widget to another by simple clicks. An example is depicted in Figure 8, for two widgets, namely, 'dafles' (a widget that we have developed for an online French dictionary – see http://ilt.kuleuven.be/blf/) and 'dpc' (a widget that we have developed for an online multilingual parallel corpus – see http://www.kuleuven-kortrijk.be/DPC/). A user looks up for the meaning of a French word in the 'dafles' widget and decides to see example sentences as well as their English translations. Therefore she clicks on the marker of one of the data items of the result list, and moves that item to the 'dpc' widget by clicking on the marker of the target form.

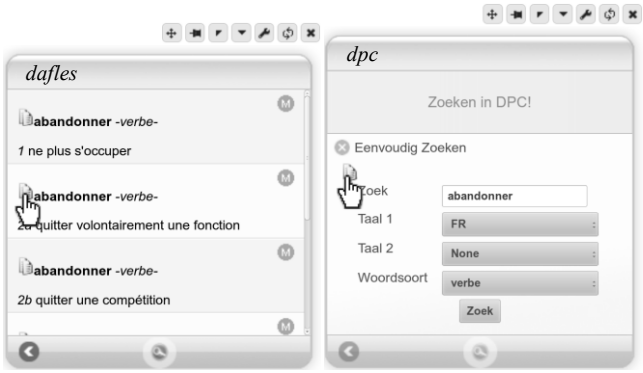

Figure 8. Data moved from one widget to another by a user.

Embedded semantics [16, 17] technologies, i.e., microformats, RDFa and eRDF, can be used for structuring content (i.e., with types and data type properties), interlinking content elements [17] (i.e., a form of linked-data - with object type properties), and embedding high-level domain semantics (e.g., class – subclass relationships) through in-content annotations. Figure 9 and Figure 10 show excerpts from annotated HTML content (with RDFa [16]) of the 'dafles' and 'dpc' widgets and extracted semantic data from them in N-Triples format. The excerpt shown in Figure 9 and Figure 10 belongs to a user-selected data item (the left-hand side of Figure 8), and a target HTML form (the right-hand side of Figure 8) respectively. The visual marking of annotated content is handled through a specific widget plug-in that we have developed.

❶ HTML: *dafles*

```
1 <span about="www.dafles.com#r1" typeof="ll:verb">
2     <span property="ll:text">abandonner</span>
3     <span property="ll:language">FR</span>
4  </span>
```

❷ N-Triple: *dafles*

```
5 <dafles:r1> <rdf:type> <ll:verb>.
6 <dafles:r1> <ll:text> "abandonner".
7 <dafles:r1> <ll:language> "FR".
```

Figure 9. Semantic data extracted from an annotated content.

In order to move a user-selected data chunk from a source widget to a target widget, a special event 'dataSelected' (i.e., copy) is introduced to inform the platform. The event also includes the user-selected data chunk. The extracted data indeed forms a RDF graph. Later, the user clicks on the marker of the target form element, and the target widget informs the platform about this special event 'formSelected (i.e., paste). The event also includes the data copied from the source widget which can be represented by another (partially) empty graph. Thus, interoperability can be achieved through graph matching. We transform the empty graph into a SPARQL query (see Figure 11), and execute it over the RDF graph along ontological reasoning. As a result, the empty graph is matched with the former RDF graph, and the values of the form elements are filled out with the data selected within the source widget. We introduce a specific name space (http://itec-research.be/ns/param) to define variable type resources in order to annotate form fields as shown in Figure 10.

❶ HTML: *dpc*

```
1  <span about="param:p" typeof="ll:word">
2     Zoek:
3    <div  rel="ll:text" resource="param:woord">
4         <input id="woord" type="text"/>
5    </div>
6     Taal1
7    <div  rel="ll:language" resource="param:taal1">
8         <select id="taal1">
9             <option value="EN">FR</option>            …
10          </select>
11       </div>                                          …
12      Woordsort
13      <div  rel="rdf:type" resource="param:woordsoort">
14         <select id="woordsoort">
15             <option value="noun">Noun<option>     …
16         </select>
17      </div>
18 </span>
```

❷ N-Triple: *dpc*

```
19 <param:p> <rdf:typeof> <ll:word>.
20 <param:p> <rdf:typeof> <param:woordsoort>.
21 <param:p> <ll:text> <param:woord>.
22 <param:p> <ll:taal> <param:taal1>.
```

Figure 10. Semantic data extracted from an annotated form.

☛ SPARQL:

```
1  SELECT ?text ?taal1 ?woordsoort
2          WHERE { ?p rdf:type ll:word.
3                  ?p rdf:type ?woordsoort.
4                  ?p ll:text ?text.
5                  ?p ll:language ?taal1.}
```

Figure 11. Form transformed into a SPARQL query.

3.2 User Behavior Mining

In the context of this paper, a *behavioral pattern* is a sequence of connected actions with control-flow dependencies (e.g., sequence, parallel, choice etc.). Each event refers to an action, and each action refers to a user executable function of a widget. Events are considered as being atomic. We investigate the use of *workflow/process mining* techniques [18] to discover user behavioral patterns from the event logs. The *α-algorithm* [19] is used for finding patterns and extracting their topologies (i.e., flow structure) and *decision point analysis*, as employed in [20], is used to find the routing criteria at decision points where routing is deterministic with respect to data in the flow.

Workflow mining approaches usually assume that there is a fully connected workflow (process) model to be discovered. In most cases, there is even an a priori prescriptive or descriptive model. In purely event-based approaches, the event log is assumed to be complete [19], that is the log is representative and a sufficiently large subset of possible behaviors is observed. The event log is subject to noise due to rare events, exceptions etc. The use of *frequency tables* is the most classical approach to deal with the noise [19]. Existing approaches are focused on the discovery of the topology (i.e., structure), and a few consider how data affects routing [20], nevertheless with a syntactic perspective. In behavior mining, there is no a priori model. And it is very unlikely that there is one single connected workflow, but rather that there are small fragments representing different activities. In order to emphasize this difference, we call these fragments behavioral patterns rather than models. Completeness (in another form), noise, and effect of data on routing are important issues which we address through the use of ontologies (see Figure 12).

❶ Given that:

```
1  widget_A REPORTS user_1 searchFor STRING 'car'
2  widget_B REPORTS user_1 searchFor STRING 'car'
3  widget_A REPORTS user_1 searchFor STRING 'come'
4  widget_C REPORTS user_1 searchFor STRING 'come'
```

☛ routing criteria cannot be learned, but one can conclude:

```
5  IF widget_A(input) THEN
6          widget_B(input) OR widget_C(input)
                    ~~~~
```

❷ Given that:

```
widget_A REPORTS user_1 searchFor Noun 'car'
widget_B REPORTS user_1 searchFor Noun 'car'
widget_A REPORTS user_1 searchFor Verb 'come'
widget_C REPORTS user_1 searchFor Verb 'come'
```

☛ one can conclude:

```
7  IF widget_A(input) THEN
8          IF TYPE_OF(input, Noun) THEN
9                  widget_B(input)
10         ELSE IF TYPE_OF(input, Verb) THEN
11                 widget_C(input)
                    ~~~~
```

❸ Given that:

```
12 widget_A HAS_FUNCTION lookforMeaning(Word)
13 widget_B HAS_FUNCTION searchImage(Noun)
```

☛ there is a possibility:

```
14 IF widget_B.searchImage (input) THEN
15         widget_A.lookforMeaning(input)
```

Figure 12. Application of ontologies for behavior mining.

In behavior mining, the completeness problem does not exist as such. In workflow mining parts of a flow structure existing in reality might be missing due to unobserved activities. In behavior mining on the contrary, the subject user constructs the reality and what exists is what we observe. We consider the completeness problem in a different form however, precisely on the basis of exploration. In order to empower users to explore and utilize the full potential of their spaces and widgets, recommending possible widgets and actions holds a crucial role. Our semantics-based approach enables such recommendations by matching semantics of widgets' inputs (i.e., semantic signatures) and outputs (i.e., content) (see situation (3) in Figure 12). The same approach is of use to tackle the noise problem. In our approach, the semantic match between actions contributes as a heuristic factor in frequency analysis. The approach enables the use of high-level semantics of the domain in decision point analysis for learning routing criteria (e.g., see situations (1) and (2) in Figure 12), for instance by utilizing ontological class types. Although class type information can be incorporated as a separate syntactic attribute, an ontology-based approach provides reasoning support such as the classification of classes.

Once behavioral patterns are discovered along their topology and routing criteria, the next step is creating a formal representation of these patterns in order to enable validation and verification, sharing, and visualization of them. For this purpose, we employ Colored Petri nets [21] (see Figure 13) by following the approaches presented in [19, 20].

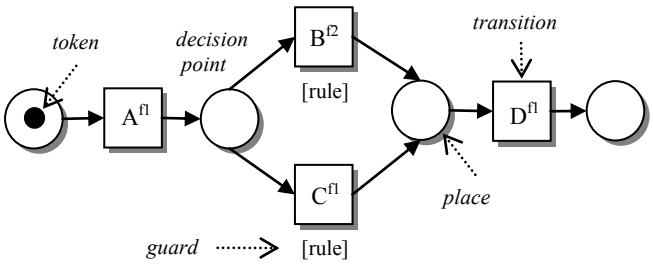

Figure 13. A Petri net: transitions refer to widget functions.

Petri nets are a graphical and mathematical modeling tool providing the ability to create, simulate and execute behavioral models; its sound mathematical model allows analysis (e.g., performance), validation, and verification of behavioral models (e.g., deadlock, liveness, reachability, boundness etc.). A Petri net is a graph in which nodes are *places* (circles) and *transitions* (rectangles). Petri nets are marked by placing *tokens* on places and a transition *fires* when all its input places (all the places with arcs to a transition) have a token. *Colored Petri nets* [21] are a type of high level Petri nets allowing association of data collections with tokens (i.e., colored, typed tokens) and describing types of token that can be located on a place (i.e., colored places). Transitions have *guards* constraining the firing of a transition with respect to incoming data. A token corresponds to data in the flow. All the in/output messages of the widget actions are modeled as colored places, and widget actions themselves are modeled as transitions with input/output places by following adaption of Colored Petri nets for service composition, e.g., [22]. Learned routing criteria are represented in terms of rules corresponding to guards in Petri nets. The work presented in [23] introduces a Petri net ontology, and can be integrated to utilize ontological reasoning and to support sharing behavioral patterns.

3.3 Platform and Architecture

The platform is composed of two primary layers (see Figure 14), namely, a *run-time system* and a *backend system*. The run-time system resides at the client (e.g., browser) and is responsible for the operational tasks and the delivery of standard platform services (e.g., preference management, see [10]) to the widget instances. The backend system resides at the server side and is responsible for the persistence and decision making.

The run-time system and backend system are composed of different components. Regarding the run-time system: (1) *Widget containers* (e.g., a HTML frame), in our context, hold widget instances in user's space and bridge communication ends of widget instances and the environment. Triggers for basic facilities, related to the presence of a widget instance in the environment such as remove, close, minimize, pin, move etc., are attached to the containers. (2) The *environment controller* manages presence related facilities, such as (absolute/relative) widget positioning, for widget instances over the widget containers and is responsible for the introduction of new sub-spaces, repositories, and widget instances (widgets from repositories or standalone widgets from the Web). (3) The *communication channel* allows bidirectional communication between a widget instance and the environment. Widget instances communicate events, preferably also preferences and data access requests to the platform, and the platform communicates data and control commands for orchestration to the widget instances through the communication channel. (4) The *run-time system core* provides standard system services to the widget instances, particularly, for preference management through (4a) the *preference management service*, for event delivery through (4b) the *event management service*, and for data access requests to the widget backend services through (4c) the *data access service* with (5) the *proxy agent*. The core coordinates orchestration through (4d) *adaptation controller* by submitting control commands to the widgets over the communication channel. The adaptation controller handles data mediation and transportation and it can utilize a light-weight (6) *client-side reasoner* for this purpose. The adaptation controller can also submit re-positioning requests (e.g., to move widgets in interplay closer) to the environment controller.

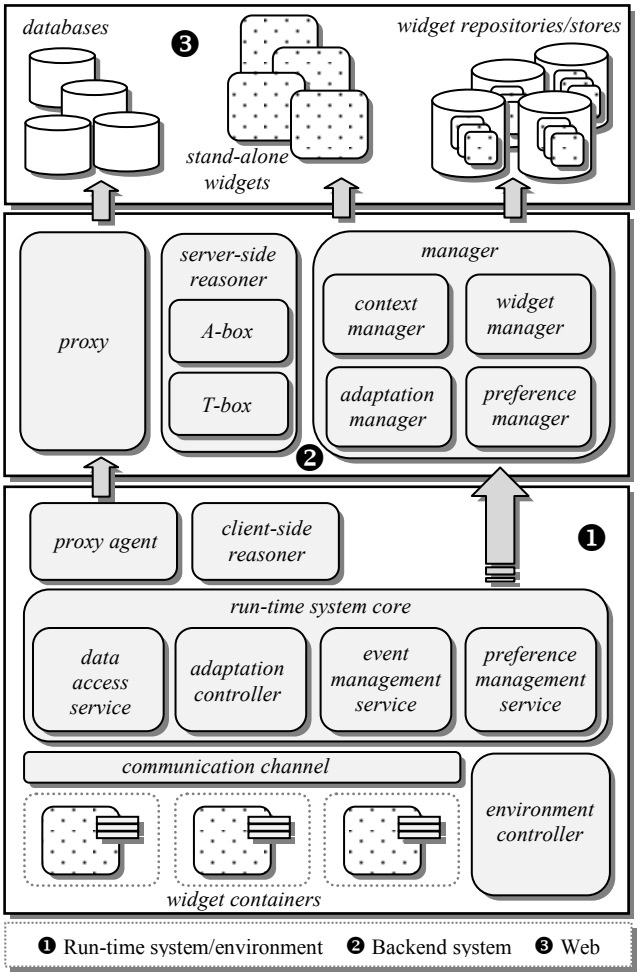

Figure 14. The platform architecture.

Regarding the backend system components, (1) the *manager* handles persistence of preferences through (1a) the *preference manager,* and the state of the environment (e.g., widgets, widget positions etc.) through (1b) the *widget manager*. The (1c) *context manager* stores event logs as well as any other contextual information for context based adaptation [24]. (1d) The *adaptation manager* decides on adaptation rules, particularly through learning behavioral patterns, and submits them to the adaptation controller. It utilizes a (2) *server-side reasoner.* (3) The *proxy* is responsible of retrieving data from external data sources upon receiving a dispatch request from the proxy agent.

4. EVALUATION AND RELATED WORK

Related work, that has been cited throughout the text, has mainly been presented in an integrative manner. In this section, the remaining body is covered with a qualitative comparison in order to evaluate our proposed strategies.

Although a considerable amount of work exists on mashups by composition due to popularity of web service composition, e.g., [2, 5-7, 25], there is limited work on mashups by orchestration which we address in what follows. Regarding widget interoperability, in the current literature, the trend is towards *inter-widget communication* (IWC) [13, 26-29] in which basically an event is delivered to relevant widgets (i.e., *unicast*, *multicast*, or *broadcast*) whenever a user performs an action inside a widget. Multicast and broadcast are the basis of IWC. In the former, widgets subscribe to particular events and/or particular widgets etc. and get notified accordingly while, in the latter, events are delivered to all widgets in a platform. In both cases, the receiving widgets decide whether and how they react to an event depending on the event type, content, etc. However, there exist some problems in IWC. First of all, since widgets have to decide on which events to react and how, widgets are overloaded with extra business logic to realize responsiveness. Secondly, responsiveness is hard to be realized. Either widgets have pre-knowledge of each other, and hence semantics of the events they deliver, or widgets exhibit responsiveness through trying to match syntactic signature of the events delivered. The former approach is not realistic because widgets are developed by different parties and in a broad, public manner. The latter is problematic in terms of its success; syntactic signatures are simply not enough for a successful identification of relevant events. The problem can be addressed through a similar approach to ours by enhancing event signatures with domain knowledge and monitoring user interaction. However, without delegating data mediation and decision tasks to the platform, it will further complicate widgets and widget development. Thirdly, since each widget acts independently, without any centralized control, it is unlikely to achieve a healthy orchestration; chaotic situations are most probably to arise when several widgets respond to the same events.

In current examples of widget-based environments, e.g., [13, 29, 30], the idea of interplay between widgets already exists, however it is either pre-designed or purely based on syntactic or semantic similarities between widgets, and behavioral patterns are not reflected. Due to that, a formal ground for mashups by orchestration is not explored. In [31], a widget-based environment (i.e., mashups by orchestration) is used for end-user programming of composite applications (i.e., mashups by composition). The work is particularly relevant since it aims at empowering end-users to program by demonstration. Each source is represented as a widget and an end-user performs a set of actions over these widgets to achieve the outcome she desires. The actions of the user are monitored and a composite application is generated respectively. The algorithm employed corresponds to a part of the α-algorithm, however a formal modeling instrument, such as Petri nets, is not utilized. Regarding architecture, the existing work is mainly repository-centric [13, 29]; Apache Wookie server (http://getwookie.org/) is notable. Wookie does not only host widgets, but also provide basic services such as inter-widget communication (over server-side communication mechanisms), preference management etc. Widgets access services that are provided by the widget server through containers in which they are placed by the server before the delivery. Such a centralized approach is inflexible and overloads the notion of repository by aggregating services and tasks, that should normally be provided by a client-side run-time system, to itself.

We have implemented a prototype to prove the applicability of our approaches. The communication channel constitutes the backbone of the platform. It is based on the *window.postmessage* method of HTML 5 allowing cross-origin communication. Run-time system services are mainly built on top of the communication channel, hereby allowing us to come up with a non-complex and generic platform and architecture. Several widgets have been developed by following the W3C widget specification [10] for language learning, hence we constructed a widget-based personal learning environment (WIPLE). Although we provide an extended platform, particularly in terms of provided run-time system services, since we use existing standards in the core of the platform, widgets we have developed are cross-platform (e.g., Opera platform - https://widgets.opera.com – enables widgets to run on desktop). A demo can be watched online from http://www.ahmetsoylu.com/pubshare/medes2011/.

5. CONCLUSION AND FUTURE WORK

In this paper, we have put a global perspective on mashups and introduced our strategies for the realization of mashups by orchestration. We have addressed interoperability, orchestration and architectural challenges by grounding their feasibility on the existing literature, standards and a prototype we have implemented. Our future work includes data collection for experimental and quantitative evaluation of our strategies.

6. ACKNOWLEDGMENTS

This paper is based on research funded by the Industrial Research Fund (IOF) and conducted within the IOF Knowledge platform 'Harnessing collective intelligence in order to make e-learning environments adaptive' (IOF KP/07/006). Partially, it is also funded by the European Community's 7th Framework Programme (IST-FP7) under grant agreement no 231396 (ROLE project).

7. REFERENCES

[1] Ankolekar, A., Krotzsch, M., Tran, T., Vrandecic, D. 2008. The two cultures: Mashing up Web 2.0 and the Semantic Web. J. Web Semant. 6, 70-75. DOI= http://dx.doi.org/10.1016/j.websem.2007.11.005.

[2] Sheth, A.P., Gomadam, K., Lathem, J. 2007. SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. IEEE Internet Comput. 11, 91-94. DOI= http://dx.doi.org/10.1109/MIC.2007.133.

[3] Baresi, L., Guinea, S. 2010. Consumer Mashups with Mashlight. In *Proceedings* of *Third European Conference, ServiceWave 2010* (Ghent, Belgium, December 13-15, 2010). Springer, Berlin, Heidelberg, 112-123. DOI= http://dx.doi.org/10.1007/978-3-642-17694-4_10.

[4] Soylu, A., Wild, F., Mödritscher, F., De Causmaecker, P. 2010. Semantic Mash-Up Personal and Pervasive Learning Environments (SMupple). In *Proceedings 6th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering, USAB 2010* (Klagenfurt, Austria, November 4-5, 2010). Springer-Verlag, Berlin, 501-504. DOI= http://dx.doi.org/10.1007/978-3-642-16607-5_37.

[5] Benslimane, D., Dustdar, S., Sheth, A. 2008. Service Mashups: The New Generation of Web Applications. IEEE Internet Comput. 12, 13-15. DOI= http://dx.doi.org/10.1109/MIC.2008.110.

[6] Tummarello, G., Cyganiak, R., Catasta, M., Danielczyk, S., Delbru, R., Decker, S. 2010. Sig.ma: Live views on the Web of Data. J. Web Semant. 8, 355-364. DOI= http://dx.doi.org/10.1016/j.websem.2010.08.003.

[7] Kopecky, J., Gomadam, K., Vitvar, T. 2008. hRESTS: An HTML Microformat for Describing RESTful Web Services. In *Proceedings of International Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2008* (Sydney, Australia, December 9-12, 2008). IEEE CS, 619-625. DOI= http://dx.doi.org/10.1109/WIIAT.2008.379.

[8] Soylu, A., Mödritscher, F., De Causmaecker, P. 2010. Utilizing Embedded Semantics for User-Driven Design of Pervasive Environments. In *Proceedings of 4th International Conference, MTSR 2010* (Alcalá de Henares, Spain, October 20-22, 2010). Springer, Berlin, Heidelberg, 63-77. DOI= http://dx.doi.org/10.1007/978-3-642-16552-8_7.

[9] Xiao, Z., Wen, S., Yu, H., Wu, Z., Chen, H., Zhang, C., Ji, Y. 2010. A new architecture of web applications-The Widget/Server architecture. In *Proceedings of 2nd IEEE International Conference on Network Infrastructure and Digital Content* (Beijing, China, September 24-26, 2010). IEEE, 866-869. DOI= http://dx.doi.org/10.1109/ICNIDC.2010.5657919.

[10] World Wide Web Consortium (W3C), Widget Interface, http://www.w3.org/TR/2011/WD-widgets-apis-20110607/, Last retrieved 2011.

[11] Oh, J., Haas, Z.J. 2010. Personal Environment Service Based on the Integration of Mobile Communications and Wireless Personal Area Networks. IEEE Commun.Mag. 48, 66-72. DOI= http://dx.doi.org/10.1109/MCOM.2010.5473866.

[12] Severance, C., Hardin, J., Whyte, A. 2008. The Coming Functionality Mash-up in Personal Learning Environments. Interact. Learn. Envir. 16, 47-62. DOI= http://dx.doi.org/10.1080/10494820701772694.

[13] Friedrich, M., Wolpers, M., Shen, R., Ullrich, C., Klamma, R., Renzel D., Richert, A., Von Der Heiden, B. 2011. Early Results of Experiments with Responsive Open Learning Environments. J. Univers. Comput. Sci. 17, 451-471.

[14] Kindberg, T., Barton, J., Morgan, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frid, M., Krishnan, V., Morris, H., Schettino, J., Serra, B., Spasojevic, M. 2002. People, Places, Things: Web Presence for the Real World. Mobile Netw. Appl. 7, 365-376. DOI= http://dx.doi.org/10.1023/A:1016591616731.

[15] Dillon, T.S., Talevski, A., Potdar, V., Chang, E. 2009. Web of Things as a Framework for Ubiquitous Intelligence and Computing. In *Proceedings of 6th International Conference, UIC 2009* (Brisbane, Australia, July 7-9, 2009). Springer Berlin, Heidelberg, 2-13. DOI= http://dx.doi.org/10.1007/978-3-642-02830-4_2.

[16] Adida, B. 2008. hGRDDL: Bridging microformats and RDFa. J. Web Semant. 6, 54-60. DOI= http://dx.doi.org/10.1016/j.websem.2007.11.006.

[17] Bizer, C., Heath, T., Berners-Lee, T. 2009. Linked Data: The Story So Far. Int. J. Semant. Web Inf. 5, 1–22. DOI= http://dx.doi.org/10.4018/jswis.2009081901.

[18] van der Aalst, W.M.P., Pesic, M., Song. M. 2010. Beyond Process Mining: From the Past to Present and Future. In *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering, CAiSE'10* (Hammamet, Tunisia, June 7-9, 2010). Springer-Verlag, Berlin, 38-52. DOI= http://dx.doi.org/10.1007/978-3-642-13094-6_5.

[19] van der Aalst, W.M.P., Weijters, T., Maruster, L. 2004. Workflow Mining: Discovering Process Models from Event Logs. IEEE T. Knowl. Data En. 16, 1128-1142. DOI= http://dx.doi.org/10.1109/TKDE.2004.47.

[20] Rozinat, A., Mans, R.S., Song, M., van der Aalst, W.M.P. 2008. Discovering colored Petri nets from event logs. Int. J. Softw. Tools Technol. Transfer 10, 57-74. DOI= http://dx.doi.org/10.1007/s10009-007-0051-0.

[21] Jensen, K., Kristensen, L.M. 2009. *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer Berlin, Heidelberg.

[22] Tan, W., Fan, Y.S., Zhou, M., Tian, Z. 2010. Data-Driven Service Composition in Enterprise SOA Solutions: A Petri Net Approach. IEEE T. Autom. Sci. Eng. 7, 686-694. DOI= http://dx.doi.org/10.1109/TASE.2009.2034016.

[23] Gasevic, D., Devedzic, V. 2006. Petri net ontology. Knowl.-based Syst. 19, 220-234. DOI= http://dx.doi.org/10.1016/j.knosys.2005.12.003.

[24] Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., Riboni, D. 2010. A survey of context modelling and reasoning techniques. Pervasive Mobile Computing 6, 161-180. DOI= http://dx.doi.org/10.1016/j.pmcj.2009.06.002.

[25] Taivalsaari, A. 2009. Mashware: The future of web applications. Technical report, Sun Microsystems.

[26] Pohja, M., 2010. Server Push for Web Applications via Instant Messaging. J. Web Eng. 9, 227-242.

[27] Wu, X., Krishnaswamy, V. 2010. Widgetizing Communication Services. In *Proceedings of 2010 IEEE International Conference on Communications, ICC* (Cape Town, South Africa, May 23-27, 2010). IEEE, 1-5. DOI= http://dx.doi.org/10.1109/ICC.2010.5502397.

[28] Sire, S., Paquier, M., Vagner, A., Bogaerts, J. 2009. A Messaging API for Inter-Widgets Communication. In *Proceedings of WWW 2009* (Madrid, Spain, April 20-24, 2009). ACM, 1115-1116. DOI= http://dx.doi.org/10.1145/1526709.1526884.

[29] Nelker, T. 2009. An Infrastructure for Intercommunication between Widgets in Personal Learning Environments. In *Proceedings of WSKS 2009* (Chania, Crete, Greece, September 16-18, 2009). Springer, Berlin, Heidelberg, 41-48. DOI= http://dx.doi.org/10.1007/978-3-642-04757-2_5.

[30] Wild F., Mödritscher, F., Sigurdarson, S. Designing for Change: Mash-Up Personal Learning Environments. eLearning Papers 9, 2008.

[31] Sinisa, S., Dejan, S., Daniel, S. 2009. Widget-Oriented Consumer Programming. Automatika 50, 252-264.